JavaScript Callbacks

❮ PreviousNext ❯

"I will call back later!"

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

Function Sequence

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

This example will end up displaying "Goodbye":

Example

```
function myFirst() {
  myDisplayer("Hello");
}

function mySecond() {
  myDisplayer("Goodbye");
}

myFirst();
mySecond();
```

Try it Yourself »

This example will end up displaying "Hello":

Example

```
function myFirst() {
  myDisplayer("Hello");
}

function mySecond() {
  myDisplayer("Goodbye");
}
```

```
mySecond();
myFirst();
```

Try it Yourself »

Sequence Control

Sometimes you would like to have better control over when to execute a function.

Suppose you want to do a calculation, and then display the result.

You could call a calculator function (`myCalculator`), save the result, and then call another function (`myDisplayer`) to display the result:

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

Try it Yourself »

Or, you could call a calculator function (`myCalculator`), and let the calculator function call the display function (`myDisplayer`):

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Try it Yourself »

The problem with the first example above, is that you have to call two functions to display the result.

The problem with the second example, is that you cannot prevent the calculator function from displaying the result.

Now it is time to bring in a callback.

JavaScript Callbacks

A callback is a function passed as an argument to another function.

Using a callback, you could call the calculator function (`myCalculator`) with a callback (`myCallback`), and let the calculator function run the callback after the calculation is finished:

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Try it Yourself »

In the example above, `myDisplayer` is a called a **callback function**.

It is passed to `myCalculator()` as an **argument**.

Note

When you pass a function as an argument, remember not to use parenthesis.

Right: myCalculator(5, 5, myDisplayer);

Wrong: ~~myCalculator(5, 5, myDisplayer())~~;

Example

```
// Create an Array
const myNumbers = [4, 1, -20, -7, 5, 9, -6];

// Call removeNeg with a callback
const posNumbers = removeNeg(myNumbers, (x) => x >= 0);

// Display Result
document.getElementById("demo").innerHTML = posNumbers;

// Keep only positive numbers
function removeNeg(numbers, callback) {
  const myArray = [];
  for (const x of numbers) {
    if (callback(x)) {
      myArray.push(x);
    }
  }
  return myArray;
}
```

Try it Yourself »

In the example above, `(x) => x >= 0` is a **callback function**.

It is passed to `removeNeg()` as an **argument**.

When to Use a Callback?

The examples above are not very exciting.

They are simplified to teach you the callback syntax.

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Asynchronous functions are covered in the next chapter.